

Circle separability queries in logarithmic time

Greg Aloupis *

Luis Barba *

Stefan Langerman *

Abstract

Let P be a set of n points in the plane. In this paper we study a new variant of the circular separability problem in which a point set P is preprocessed so that one can quickly answer queries of the following form: Given a geometric object Q , report the minimum circle containing P and excluding Q . Our data structure can be constructed in $O(n \log n)$ time using $O(n)$ space, and can be used to answer the query when Q is either a circle or a convex m -gon in $O(\log n)$ or $O(\log n + \log m)$ time, respectively.

1 Introduction

The planar separability problem consists of constructing, if possible, a boundary that separates the plane into two components such that two given sets of geometric objects become isolated. Typically this boundary is a single curve such as a line, circle or simple polygon, meaning that each component of the plane is connected. Often there is an additional objective of minimizing some feature of the boundary, for instance the radius of a circle.

Probably the most classic instance of this problem is to separate two given point sets with a circle (or a line, which is equivalent to an infinitely large circle). This problem was developed because of the several applications it has in pattern recognition and image processing [12, 13]. Circle separability has been extensively studied and usually two variants are considered: The decision problem, where the existence of the separating circle is tested, and the Min-Max problem, where the minimum or maximum separating circle is to be determined.

A separating line can be found, if it exists, using linear programming. In the plane this takes linear time by Megiddo's algorithm [15]. For circle separability, O'Rourke, Kosaraju and Megiddo [16] gave a linear-time algorithm for the decision problem (in fact spherical separability in any fixed dimension), improving earlier bounds [5, 13]. They also gave an $O(n \log n)$ algorithm for finding the largest separating circle and a linear time algorithm for finding the minimum separating circle between any two finite point sets. Extending these ideas, Boissonat et al. [6] gave a linear-time algorithm to report the smallest separating circle for two simple polygons, if it exists.

The separability problem has also been studied when two point sets are to be isolated by the boundary of a simple polygon. Edelsbrunner and Preparata [11] gave an $O(n \log n)$ -time algorithm to find a separating convex polygon with minimum number of vertices. This was shown to be optimal if the optimal separator has linear size. They also gave a $O(kn)$ algorithm to find a convex polygonal separator with either k or $k+1$ vertices, where k is the size of the optimal solution.

Aggarwal et al. [1] gave an $O(n \log k)$ -time algorithm to find the separating (convex) polygon with fewest vertices, between two nested convex polygons. Again, k is the size of the optimal separator.

*Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium

Das and Joseph extended the problem to higher dimensions and proved that the problem of computing a separating polyhedron, having the minimum number of faces, for two nested convex polyhedra is NP-complete [7]. Suri and O'Rourke [20] gave a quadratic time algorithm to find a polygonal separator of two simple, not necessarily convex, nested polygons. This was improved to $O(n \log n)$ time by Wang [23]. Finally, Wang and Chan [24] gave an $O(n \log n)$ algorithm to find a minimal polygonal separator between two arbitrary, not necessarily nested, simple polygons.

In some situations, the given geometric objects and the separating boundary are constrained to lie within some subset of the plane, such as a simple polygon. For instance, Demaine et al. [8] studied the separation of point sets by chords and geodesic paths inside polygons.

The online version of the separability problems, in which a preprocess is allowed to answer separability queries, is of special interest when a lot of queries are to be processed. This is the case in areas like geometric modeling involving objects in motion, where collision detection or largest empty space recognition queries are extensively used. Therefore, the online variants of some separability problems have been studied as well: Augustine et al [3] show how to preprocess a point set P , so that the largest circle, isolating P from a query point, can be found in logarithmic time. They also obtain this query time when P represents the boundary of a simple polygon.

For the line separability problem, Edelsbrunner show that a point set P can be preprocessed in $O(n \log n)$ time, so that a separating line between P and a query convex m -gon Q can be computed in $O(\log n + \log m)$ time [10]. In 3D space, Dobkin and Kirkpatrick show that two convex polyhedra can be preprocessed in linear time, so that a separating plane, if any exists, can be computed in $O(\log n \cdot \log m)$ time, given any orientation and position of both polyhedra. In this case, n and m represent the size of each polyhedra.

In this paper we show that a point set P on n points can be preprocessed in $O(n \log n)$ time, using $O(n)$ space, so that for any given convex m -gon Q , we can find the smallest circle enclosing P and excluding Q in $O(\log n + \log m)$ time. This improves the $O(\log n \cdot \log m)$ bound presented in [4], which is described in this paper as well.

2 Preliminaries

Let P be a set of n points in the plane and let Q be a convex m -gon. We say that every circle containing P is a P -circle. We also say that a circle C separates P from Q , or simply that C is a *separating circle*, if C is a P -circle and its interior does not intersect Q . Likewise, a *separating line* is a straight line leaving the interiors of P and Q in different halfplanes.

Let \mathbf{C}' denote the minimum separating circle and let \mathbf{c}' be its center. Note that \mathbf{C}' always passes through at least two points of P , since otherwise a smaller separating circle can always be found. In fact \mathbf{c}' must lie on an edge of the farthest-point Voronoi diagram $\mathcal{V}(P)$, which is a tree with leaves at infinity [17]. For each point p of P , let $R(p)$ be the farthest-point Voronoi region associated with p .

Let C_P be the minimum enclosing circle of P . If C_P is constrained by three points of P then its center, c_P , is at a vertex of $\mathcal{V}(P)$. Otherwise C_P is constrained by exactly two points of P (forming its diameter), in which case c_P is on the interior of an edge of $\mathcal{V}(P)$. If that is the case, we will insert c_P into $\mathcal{V}(P)$ by splitting the edge where it belongs. Thus, we can think of $\mathcal{V}(P)$ as a rooted tree on c_P . For any given point x on $\mathcal{V}(P)$ there is a unique path along $\mathcal{V}(P)$ joining c_P with x , throughout this paper we will denote this path by π_x .

Given any point y in the plane, let $C(y)$ be the minimum P -circle with center on y and let $\rho(y)$ be the radius of $C(y)$. Note that if y belongs to $R(p)$ for some point p of P , then $\rho(y)$ is given by $d(y, p)$, where $d(\cdot, \cdot)$ denotes the Euclidian distance between any two geometric objects. Finally, we say that y is a *separating point* if $C(y)$ is a separating circle.

3 Properties of the minimum separating circle

In this section we describe some properties of \mathbf{C}' , and the relationship between \mathbf{c}' and the farthest-point Voronoi diagram. These properties are not new. In fact most of the results in this section are either proved, stated, or assumed in [4].

Let $\text{CH}(P)$ denote the convex hull of P . We assume that the interiors of Q and $\text{CH}(P)$ are disjoint, otherwise there is no separating circle. Also, if Q and C_P have disjoint interiors, then C_P is trivially the minimum separating circle.

The following useful property of the farthest-point Voronoi diagram was mentioned in [4]. We provide a short proof.

Proposition 3.1. *Let x be a point on $\mathcal{V}(P)$. The function ρ is monotonically increasing along the path π_x starting at c_P .*

Proof. Let u, v be two vertices of $\mathcal{V}(P)$. In [18], they proved that if u is an ancestor of v , then $\rho(u) < \rho(v)$. It only remains to prove that ρ is also monotonically increasing along every edge of π_x . Let e be an edge of π_x contained in the bisector of the points p, p' of P , and consider a point y moving along e . Note that as $d = d(y, p) = d(y, p')$ increases, the radius of $C(y)$ also increases since y lies on $R(p) \cap R(p')$. Thus, by moving the point y we obtain that $\rho(y)$ increases monotonically on every edge along π_x . \square

Observation 3.2. *Every P -circle contained in a separating circle is also a separating circle.*

The following is stated in [4]. We provide a brief proof.

Proposition 3.3. *Let x and y be two points in \mathbb{R}^2 . If z is a point contained in the segment $[x, y]$, then $C(z) \subseteq C(x) \cup C(y)$.*

Proof. Let a and b be the two points of intersection between $C(x)$ and $C(y)$. Let $r = d(a, z) = d(b, z)$ and let $C_r(z)$ be the circle with center on z and radius r ; see Figure 1.

It is clear that $C(x) \cap C(y) \subseteq C_r(z)$ and since $P \subset C(x) \cap C(y)$, we infer that $C_r(z)$ is a P -circle, therefore $C(z) \subseteq C_r(z)$. Furthermore, since $C_r(z) \subseteq C(x) \cup C(y)$ by construction, we conclude that $C(z) \subseteq C(x) \cup C(y)$. \square

Observation 3.2 and Proposition 3.3 imply that if $C(x)$ and $C(y)$ are both separating circles, then for every $z \in [x, y]$, $C(z)$ is also a separating circle. This comes from the fact that $C(z)$ is contained in $C(x) \cup C(y)$ and Q lies outside of this union. Furthermore, this implies that the minimum separating circle is unique. Assume otherwise that C and C' are both minimum separating circles with centers c and c' , respectively. Thus, for every point z in the open segment (c, c') , $C(z)$ is also a separating circle contained in $C \cup C'$. This means that $C(z)$ has a smaller radius than C and C' which would be a contradiction.

The following is also demonstrated in [4], with a reworded proof.

Lemma 3.4. *Let x and y be two separating points on $\mathcal{V}(P)$. If z is the lowest common ancestor of x and y in the rooted tree $\mathcal{V}(P)$, then $C(z)$ is a separating circle; moreover $\rho(z) \leq \min\{\rho(x), \rho(y)\}$.*

Proof. Since z is a common ancestor of x and y , we infer from Proposition 3.1 that $\rho(z) \leq \min\{\rho(x), \rho(y)\}$. What remains is to prove that $C(z)$ is a separating circle. Suppose that $y \notin \pi_x$ and $x \notin \pi_y$, otherwise the result follows trivially since y or x will coincide with z . So, let e_x (resp. e_y) be the edge incident to z , on the path joining z with x (resp. y). Consider the radial order of the edges incident to z lying between e_x and e_y . Let e' be the edge consecutive to e_x in

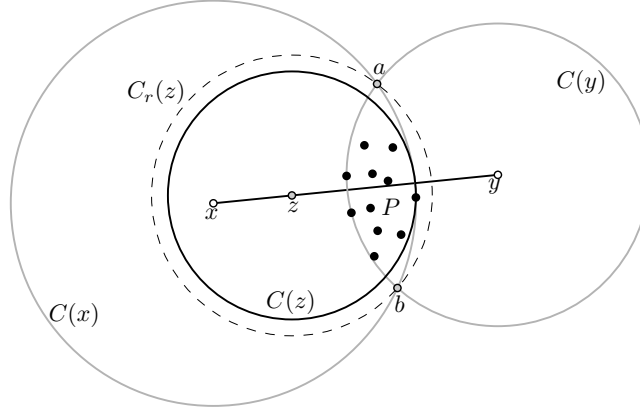


Figure 1: For every point $z \in [x, y]$ the circle $C(z)$ lies in the union of $C(x)$ and $C(y)$.

this order. Therefore, e_x, e' and z all belong to the boundary of some Voronoi region $R(p)$; see Figure 2.

Let $\ell_{z,p}$ be the line through z and p . By the definition of $\mathcal{V}(P)$, $\ell_{z,p}$ intersects the boundary of $R(p)$ only at the point z , and it separates x and y . Thus the intersection point z' between $[x, y]$ and $\ell_{z,p}$ belongs to $R(p)$. Recall that Proposition 3.3 implies that $C(z')$ is a separating circle. Trivially, $C(z) \subseteq C(z')$ since the latter is obtained by expanding $C(z)$ while anchoring it to p . Since $C(z)$ is a P -circle contained inside a separating circle, by Observation 3.2 $C(z)$ is also a separating circle. \square

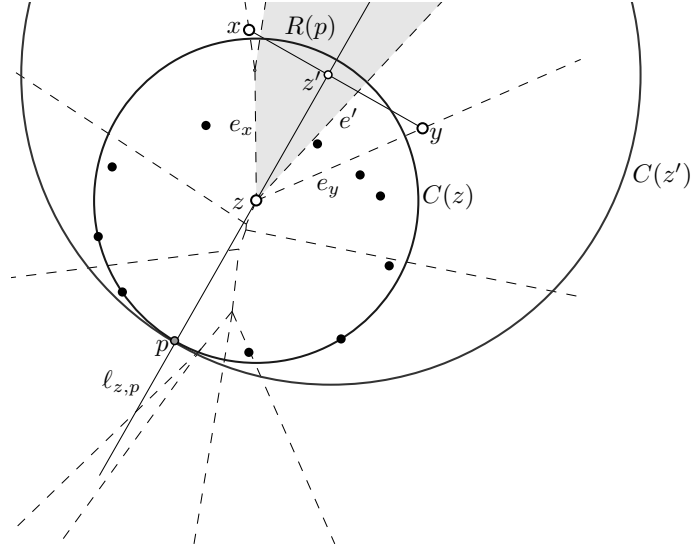


Figure 2: Illustration for Lemma 3.4.

The following is also proved in [4].

Theorem 3.5. *Let s be a point on $\mathcal{V}(P)$. If s is a separating point, then \mathbf{c}' belongs to π_s .*

Proof. Proceed by contradiction and assume that \mathbf{c}' does not belong to π_s . Let z be the lowest common ancestor of \mathbf{c}' and s . Note that z lies in π_s and therefore $z \neq \mathbf{c}'$. In this case, Lemma 3.4, in conjunction with Proposition 3.1, imply that $C(z)$ is a separating circle with $\rho(z) < \rho(\mathbf{c}')$ which is a contradiction. \square

Given a separating point s , we claim that if we move a point y continuously from s towards c_P on π_s , then $C(y)$ will shrink and approach Q , becoming tangent to it for the first time when y reaches \mathbf{c}' . To prove this claim in Lemma 3.8, we introduce the following notation.

Let x be a point lying on an edge e of $\mathcal{V}(P)$ such that e lies on the bisector of $p, p' \in P$. Let $C^-(x)$ and $C^+(x)$ be the two closed convex regions obtained by splitting the disk $C(x)$ with the segment $[p, p']$. Assume that x is contained in $C^-(x)$.

Observation 3.6. *Let x, y be two points lying on an edge e of $\mathcal{V}(P)$. If $\rho(x) > \rho(y)$, then $C^+(x) \subset C^+(y)$ and $C^-(y) \subset C^-(x)$.*

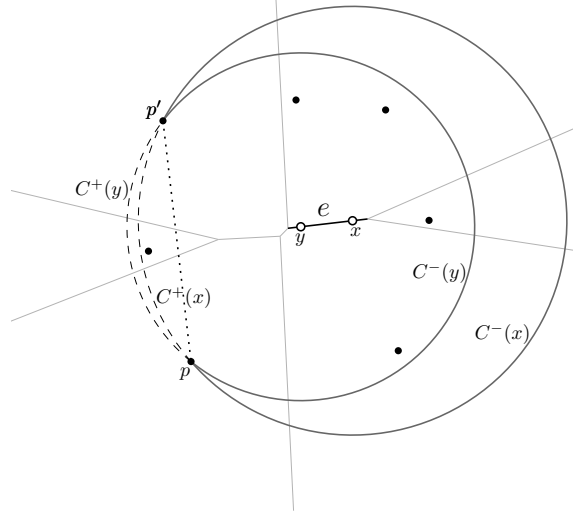


Figure 3: The result of Observation 3.6 when $\rho(x) > \rho(y)$.

Using this simple observation that can be seen in Figure 3, we obtain the following generalization.

Proposition 3.7. *Let s be a point on $\mathcal{V}(P)$ and let x and y be two points on π_s . If $\rho(x) > \rho(y)$, then $C^+(x) \subset C^+(y)$ and $C^-(y) \subset C^-(x)$.*

Proof. Note that if x and y lie on the same edge, then the result holds by Observation 3.6. If they are on different edges, we consider the path $\Phi = (x, v_0, \dots, v_k, y)$ contained in π_s joining x and y , such that v_i is a vertex of $\mathcal{V}(P)$, $i \in \{0, \dots, k\}$. Thus, Observation 3.6 and Proposition 3.1 imply that $C^+(x) \subset C^+(v_0) \subset \dots \subset C^+(v_k) \subset C^+(y)$ and that $C^-(y) \subset C^-(v_k) \subset \dots \subset C^-(v_0) \subset C^-(x)$. \square

Note that $\mathbf{C}' = C(\mathbf{c}')$ must intersect the boundary of Q . Otherwise, we could always push \mathbf{c}' closer to the root on $\mathcal{V}(P)$, while keeping it as a separating point. Furthermore, since Q is

convex and \mathbf{C}' contains no point of Q in its interior, the intersection consists on only one point. From now on we refer to ϕ' as the tangency point between \mathbf{C}' and Q .

We claim that ϕ' lies on the boundary of $C^+(\mathbf{c}')$. Assume to the opposite that ϕ' lies on $C^-(\mathbf{c}')$. Let $\varepsilon > 0$ and let c_ε be the point obtained by moving \mathbf{c}' a ε distance towards to c_P on $\mathcal{V}(P)$. Note that by Proposition 3.1, $\rho(c_\varepsilon) < \rho(\mathbf{c}')$. In addition, Proposition 3.7 implies that $C^-(c_\varepsilon) \subset C^-(\mathbf{c}')$. Since we assumed that ϕ' lies on the boundary of $C^-(\mathbf{c}')$, we conclude that ϕ' does not belong to $C(c_\varepsilon)$. This implies that, for ε sufficiently small, $C(c_\varepsilon)$ is a separating circle which is a contradiction to the minimality of \mathbf{C}' .

Lemma 3.8. *Let s be a separating point. If x is a point lying on π_s , then $C(x)$ is a separating circle if and only if $\rho(x) \geq \rho(\mathbf{c}')$. Moreover, \mathbf{C}' is the only separating circle that intersects Q .*

Proof. Let s be a separating point. We know by Theorem 3.5 that \mathbf{c}' belongs to π_s . Let x_1 and x_2 be two points on π_s such that $\rho(x_1) < \rho(\mathbf{c}')$ and $\rho(\mathbf{c}') < \rho(x_2)$. Proposition 3.7 implies that $C^+(\mathbf{c}') \subset C^+(x_1)$ and since ϕ' belongs to the boundary of $C^+(\mathbf{c}')$, we conclude $C(x_1)$ contains ϕ' in its interior. Therefore $C(x_1)$ is not a separating circle.

On the other hand, $C(x_2)$ contains no point of Q . Otherwise, let $q \in Q$ be a point lying in $C(x_2)$. Two cases arise: Either q belongs to $C^-(x_2)$ or q belongs to $C^+(x_2)$. In the former case, since $\rho(s) > \rho(x_2)$, $q \in C^-(x_2) \subset C^-(s)$ — a contradiction since $C(s)$ is a separating circle. In the latter case, since $\rho(x_2) > \rho(\mathbf{c}')$, Proposition 3.7 would imply that q belongs to the interior of \mathbf{C}' which would also be a contradiction. \square

4 The algorithm

The basis of our algorithm is to find a separating point s and from there, perform a binary search on π_s to find a separating circle tangent to Q with center on this path. This is a rather intuitive solution, which was also used in [4]. However we use some additional geometric properties to reduce the time complexity.

4.1 Preprocessing

We first compute $\mathcal{V}(P)$ and c_P in $O(n \log n)$ time [19]. $\mathcal{V}(P)$ can be stored as a binary tree with n (unbounded) leaves, so that every edge and every vertex of the tree has a set of pointers to the vertices of P defining it. Every Voronoi region is stored as a convex polygon and every vertex p of P has a pointer to $R(p)$. If c_P is not a vertex of $\mathcal{V}(P)$, we split the edge that it belongs to.

We want our data structure to support binary search queries on any possible path π_s of $\mathcal{V}(P)$. Thus, to guide the binary search we would like to have an oracle that answers queries of the following form: Given a vertex v of π_s , decide if \mathbf{c}' lies either between c_P and v or between v and s in π_s . By Lemma 3.8, we only need to decide if $C(v)$ is a separating circle.

We will use an operation on the vertices of $\mathcal{V}(P)$ called `FINDPOINTBETWEEN` with the following properties. Given two vertices u, v in π_s , `FINDPOINTBETWEEN(u, v)` returns a vertex z that splits the path on π_s joining u and v into two subpaths. Moreover, if we use our oracle to discard one of the subpaths and to proceed recursively on the other, then we want `FINDPOINTBETWEEN(u, v)` to guarantee that this recursive process ends after $O(\log n)$ steps. That is, after $O(\log n)$ iterations, the search interval becomes only an edge of π_s containing \mathbf{c}' .

A data structure build on top of $\mathcal{V}(P)$ that support this operation was presented in [18]. This data structure can be constructed in $O(n)$ time and uses linear space by storing a constant number of pointers on each vertex of $\mathcal{V}(P)$.

4.2 Searching for \mathbf{c}' on the tree

Recall that if C_P is a separating circle then it is a trivial solution. Since Q is a convex m -gon, this can be checked easily in $O(\log m)$ time [10]. Thus we will assume that C_P is not the minimum separating circle, which implies that C_P intersects Q .

To determine the position of \mathbf{c}' on $\mathcal{V}(P)$, we first find a separating point s and then search for \mathbf{c}' on π_s using our data structure.

To find s , we construct a separating line L between P and Q . This can be done in $O(\log n + \log m)$ time [10]. Let p_L be the point of P closest to L and assume that no other point in P lies at the same distance; otherwise rotate L slightly. Let L_\perp be the perpendicular to L that contains p_L and let s be the intersection of L_\perp with the boundary of $R(p_L)$; see Figure 4. We know that L_\perp intersects $R(p_L)$ because L can be considered as a P -circle, containing only p_L , with center at infinity on L_\perp .

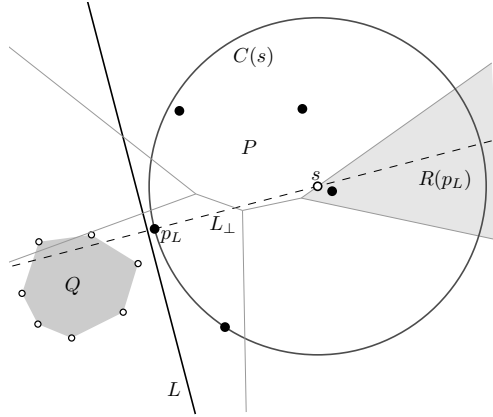


Figure 4: Construction of s .

Since s is on the boundary of $R(p_L)$, $C(s)$ passes through p_L . Furthermore $C(s)$ is contained in the same halfplane defined by L that contains P . So $C(s)$ is a separating circle. Assume that s lies on the edge \overline{xy} of $\mathcal{V}(P)$ with $\rho(x) > \rho(y)$ and let $\pi_s = (u_0 = s, u_1 = y, \dots, u_r = c_P)$ be the path of length $r + 1$ joining s with c_P in $\mathcal{V}(P)$. Theorem 3.5 implies that \mathbf{c}' lies on π_s .

It is possible to use our data structure to perform a binary search on the vertices of π_s , computing, at each vertex v , the distance to Q and the radius of $C(v)$. This way we could determine if $C(v)$ is a separating (or intersecting) circle. However, this approach involves computing the distance to Q at each step in $O(\log m)$ time, and thus takes $O(\log n \cdot \log m)$ time. This was the algorithm given in [4].

It is worth noting that if the distance to our query object can be computed in $O(1)$ time, then the described algorithm takes $O(\log n)$ time. This is the case when instead of being an m -gon, Q is either a circle or a point. The next result follows.

Observation 4.1. *After preprocessing a set P of n points in $O(n \log n)$ time, the minimum separating circle between P and any given circle or point can be found in $O(\log n)$ time.*

In the case of Q being a convex m -gon, an improvement from the $O(\log n \log m)$ time algorithm can be obtained by strongly using the convexity of Q .

To determine if some point v on π_s is a separating point, it is not always necessary to compute the distance between v and Q . One can first test, in constant time, if $C(v)$ intersects a separating line tangent to Q . If it does not intersect it, then $C(v)$ would be a separating circle and we can proceed with the binary search. Otherwise, we can try to compute a new separating line tangent to Q not intersecting $C(v)$. The advantage of this is that while doing so, we reduce the portion of Q that we need to consider in the future. This is done as follows.

Compute the two internal tangents L, L' between the convex hull of P and Q in $O(\log n + \log m)$ time. The techniques to construct these tangents are shown in Chapter 4 of [21].

Let q and q' be the respective tangency points of L and L' with the boundary of Q . We will consider the clockwise polygonal chain $\varphi = [q = q_0, \dots, q_k = q']$ joining q and q' as in Figure 5.

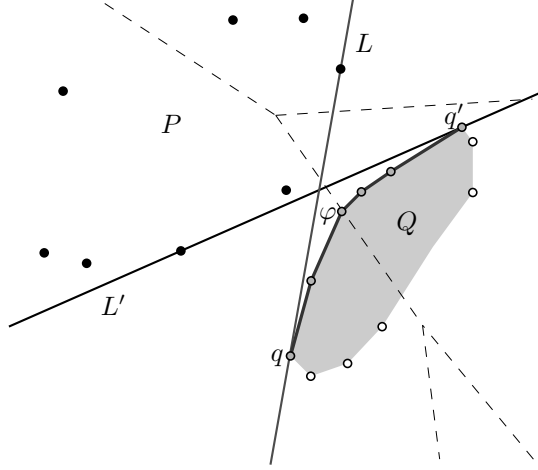


Figure 5: The construction of φ .

Recall that ϕ' denotes the intersection point between \mathbf{C}' and the boundary of Q and note that the tangent line to \mathbf{C}' at ϕ' is a separating line. Therefore, ϕ' must lie on an edge of φ since no separating line passes through any other boundary point of Q .

If $q = q'$, then $\phi' = q$ and thus, we can forget about Q and compute the minimum separating circle between P and q . As mentioned previously, by Observation 4.1 this takes $O(\log n)$ time.

Assume from now on that $q \neq q'$, as shown in Figure 5. For each edge $e_i = q_i q_{i+1}$ ($0 \leq i \leq k-1$) of φ , let ℓ_i be the line extending that edge. By construction, we know that each ℓ_i separates P and Q . We say that a point x on ℓ_i but not in e_i lies to the left of e_i if it is closer to q_i , or to the right if it is closer to q_{i+1} .

Our algorithm will essentially perform two parallel binary searches, the first one on π_s and the second one on φ , such that at each step we discard either a section of π_s or a linear fraction of φ . As we search on π_s , every time we find a separating circle, we move towards c_P . When we confirm that a P -circle intersects Q , we move away from c_P .

As mentioned, we attempt to confirm if the current vertex v being analyzed corresponds to a separating point. To do that, we compare $C(v)$ to some separating line ℓ_i for intersection, i.e., in constant time. If $C(v)$ is a separating circle, we instantly discard the section of the path lying below v on $\mathcal{V}(P)$. If $C(v)$ does intersect ℓ_i , we make a quick attempt to check if $C(v)$ intersects Q by comparing $C(v)$ and the edge e_i for intersection. If so, v is not a separating point and we can proceed with the binary search on π_s . Otherwise, the intersection of $C(v)$ with ℓ_i lies either to the left or to the right of e_i . However, in this case we are not able to quickly conclude whether

$C(v)$ intersects Q or not. Thus, we temporarily suspend the binary search on $\mathcal{V}(P)$ and focus on $C(v)$, using it to eliminate half of φ . Specifically, the fact that $C(v)$ intersects ℓ_i to one side of e_i (right or left) tells us that no future P -circle on our search will intersect ℓ_i to the other side of e_i . This implicitly discards half of φ from future consideration, and is discussed in more detail in the Theorem that follows. Thus, in constant time, we manage to remove a section of the path π_s , or half of φ , which will give the desired time bound. The entire process is detailed in Algorithm 1.

Algorithm 1 Given $\pi_s = (u_0 = s, u_1 = y, \dots, u_r = c_P)$ and $\varphi = [q = q_0, \dots, q_k = q']$, find the edge of π_s containing \mathbf{c}'

```

1: Define the endpoints of the subpath of  $\pi_s$  containing  $\mathbf{c}'$ ,  $u \leftarrow s, v \leftarrow c_P$ 
2: Define the initial search interval on  $\varphi$ ,  $a \leftarrow 0, b \leftarrow k$ 
3: if  $u$  and  $v$  are consecutive vertices and  $b = a + 1$  then
4:   End and return the segment  $S = [u, v]$  and the segment  $H = [q_a, q_b]$ 
5: end if
6: Let  $z \leftarrow \text{FINDPOINTBETWEEN}(u, v)$ ,  $j \leftarrow \lfloor \frac{a+b}{2} \rfloor$ 
7: Let  $e_j \leftarrow \overline{q_j q_{j+1}}$  and let  $\ell_j$  be the line extending  $e_j$ 
8: if  $b > a + 1$  then
9:   Compute  $\rho(z)$  and let  $\delta \leftarrow d(z, \ell_j)$ ,  $\Delta \leftarrow d(z, e_j)$ 
10: else
11:   Compute  $\rho(z)$  and let  $\delta \leftarrow d(z, e_j)$ ,  $\Delta \leftarrow d(z, e_j)$ 
12: end if
13: if  $\rho(z) \leq \delta$ , that is  $C(z)$  is a separating circle then
14:   Move forward on  $\pi_s$ ,  $u \leftarrow z$  and return to step 3
15: else
16:   if  $\rho(z) > \Delta$ , that is if  $C(z)$  is not a separating circle then
17:     Move backward on  $\pi_s$ ,  $v \leftarrow z$  and return to step 3
18:   else
19:     if  $C(z)$  intersects  $\ell_j$  to the left of  $e_j$  then
20:       We discard the polygonal chain to the right of  $e_j$ ,  $b \leftarrow \max\{j, a + 1\}$ 
21:     else
22:       We discard the polygonal chain to the left of  $e_j$ ,  $a \leftarrow j$ 
23:     end if
24:     Return to step 3
25:   end if
26: end if

```

Theorem 4.2. *Algorithm 1 finds the edge of π_s containing \mathbf{c}' in $O(\log n + \log m)$ time.*

Proof. Our algorithm maintains two invariants. The first is that $C(u)$ is never a separating circle and $C(v)$ is always a separating circle. To begin with, $C(u) = C(s)$ is a separating circle, while $C(v) = C_P$ is not a separating circle. If either of these assumptions does not hold, the problem is solved trivially, without resorting to this algorithm. Changes to u and v occur in steps 14 or 17, and in both cases the invariant is preserved. Thus, \mathbf{c}' always lies on the path joining u with v .

The second invariant is that ϕ' , the tangency point between \mathbf{C}' and Q , always lies on the clockwise path joining q_a with q_b along φ . We already explained that the invariant holds when $a = 0$ and $b = k$, corresponding to the inner tangents supporting P and Q . Thus we only need to look at steps 20 and 22, where a and b are redefined.

We will analyze step 20, however step 22 is analogous. In step 20 we know that $C(z)$ intersects ℓ_j to the left of e_j and that e_j does not intersect $C(z)$. We claim that for every point w lying on an edge of π_s , if $C(w)$ is a separating circle that intersects ℓ_j , then it intersects it to the left of e_j . Note that if our claim is true, we can forget about the polygonal chain lying to the right of e_j since no separating circle will intersect it. To prove our claim, suppose that there is a point w on π_s , such that $C(w)$ is a separating circle and $C(w)$ intersects ℓ_j to the right of e_j . Let x and x' be two points on the intersection of ℓ_j with $C(w)$ and $C(z)$, respectively. Suppose first that $\rho(w) < \rho(z)$ and recall that by Proposition 3.7, since x' lies on $C^+(z) \subset C^+(w)$, x' lies in $C(w)$. Thus, both x and x' belong to $C(w)$ which by convexity implies that e_j is contained in $C(w)$. Therefore $C(w)$ is not a separating circle which is a contradiction. Analogously, if $\rho(w) > \rho(z)$, then e_j is contained in $C(z)$ which is directly a contradiction since we assumed the opposite; our claim holds.

Note that in each iteration of the algorithm, a, b, u or v are redefined so that either a linear fraction of φ is discarded, or a part of π_s is discarded and a new call to `FINDPOINTBETWEEN` is performed. Recall that our data structure guarantees that $O(\log n)$ calls to `FINDPOINTBETWEEN` are sufficient to reduce the search interval in π_s to an edge [18]. Thus, the algorithm finishes in $O(\log n + \log m)$ iterations.

One extra detail needs to be considered when $b = a+1$. In this case only one edge $e = [q_a, q_{a+1}]$ remains from φ , and ϕ' lies on that edge. Thus, if the line ℓ extending e intersects $C(z)$ but e does not, then either step 20 or 22 is executed. However, nothing will change in these steps and the algorithm will loop. In order to avoid that, we check in step 8 if only one edge e of φ remains. If this is the case, we know by our invariant that ϕ' belongs to that edge and therefore we continue the search computing the distance to e instead of computing the distance to the line extending it. This way, the search on φ stops but it continues on π_s until the edge of $\mathcal{V}(P)$ containing \mathbf{c}' is found.

Since we ensured that every edge in $\mathcal{V}(P)$ has pointers to the points in P that defined it, every step in the algorithm can be executed in $O(1)$ time. Thus, we conclude that Algorithm 1 finishes in $O(\log n + \log m)$ time.

Since both invariants are preserved during the execution, Lemma 3.8 implies that the algorithm returns segments $[u, v]$ from π_s containing \mathbf{c}' , and $[q_a, q_b]$ from φ containing ϕ' . \square

From the output of Algorithm 1 it is trivial to obtain \mathbf{c}' in constant time, so we conclude the following.

Corollary 4.3. *After preprocessing a set P of n points in $O(n \log n)$ time, the minimum separating circle between P and any query convex m -gon can be found in $O(\log n + \log m)$ time.*

References

- [1] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri and C.K. Yap, Finding minimal convex nested polygons, *Proceedings of the first annual symposium on Computational geometry* (1985), 296–304.
- [2] A. Aggarwal, L.J. Guibas, J. Saxe and P.W. Shor, A linear-time algorithm for computing the Voronoi diagram of a convex polygon, *Discrete Comput. Geom.* (1989), 4:591–604.
- [3] J. Augustine, S. Das, A. Maheshwari, S.C. Nandy, S. Roy and S. Sarvattomananda, Localized geometric query problems, *arXiv:1111.2918* (2011)

- [4] L. Barba and J. Urrutia, Dynamic circle separability between convex polygons *XIV Spanish Meeting on Computational Geometry* (2011), 43–46.
- [5] B. K. Bhattacharya, Circular separability of planar point sets, *Computational Morphology* (1988), 25–39.
- [6] J-D. Boissonat, J. Czyzowicz, O. Devillers and M. Yvinec, Circular separability of polygons, *Algorithmica* (2001), 30:67–82.
- [7] G. Das and D. Joseph, The complexity of minimum convex nested polyhedra, *In Proc. 2nd Canad. Conf. Comput. Geom.* (1990), 296–301.
- [8] E. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, H. Meijer, M. Overmars and S. Whitesides, Separating point sets in polygonal environments, *Proc. 20th Annual ACM Symposium on Computational Geometry (SoCG)* (2004), 10–16.
- [9] Dobkin, David and Kirkpatrick, David, Determining the separation of preprocessed polyhedra A unified approach, *Lecture Notes in Computer Science*, (1990), Volume 443, 400–413.
- [10] H. Edelsbrunner, Computing the extreme distances between two convex polygons, *Journal of Algorithms* (1985), 6:213–224.
- [11] H. Edelsbrunner and F.P. Preparata, Minimum polygonal separation, *Inform. Comput.* (1988), 77:218–232.
- [12] S. Fisk, Separating points by circles and the recognition of digital discs, *IEEE Trans. Pattern Anal. Mach. Intell.* (1986), 8(4):554–556.
- [13] C. E. Kim and T. A. Anderson, Digital disks and a digital compactness measure, *STOC '84: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, (1984), 117–124.
- [14] S.R. Lay, On separation by spherical surfaces, *Amer. Math. Monthly* **1** (1971), 78:1112–1113.
- [15] N. Megiddo, Linear programming in linear time when the dimension is fixed, *J. ACM* (1984), 31:114–127.
- [16] J. O'Rourke, S.R. Kosaraju and N. Megiddo, Computing circular separability, *Discrete and Computational Geometry* **1** (1986), 105–113.
- [17] Mark de Berg and Marc van Kreveld and Mark Overmars and Otfried Cheong Schwarzkopf, *Computational geometry: Algorithms and applications*, Springer, (2000), Second Edition.
- [18] S. Roy, A. Karmakar, S. Das and S.C. Nandy, Constrained minimum enclosing circle with center on a query line segment, *Computational Geometry* (2009), 42:632–638.
- [19] M.I. Shamos and D. Hoey, Closest point problems, *in Proceedings of 16th Annual IEEE Symposium on Foundations of Computer Science*, (1975), 151–162.
- [20] S. Suri and J. O'Rourke, Finding minimal nested polygons, *Proc. 23rd Allerton Conf. Commun. Control Comput.* (1985), 470–479.
- [21] Preparata, Franco P. and Shamos, Michael I., *Computational geometry: an introduction*, 1985, Springer-Verlag New York, Inc.

- [22] G.T. Toussaint, Solving geometric problems with the rotating calipers, *Proceedings of IEEE MELECON'83, Athens, Greece* (1983).
- [23] C. Wang, Finding minimal nested polygons, *BIT Numerical Mathematics* (1991) 31(2):230–236.
- [24] C. Wang and E.P.F. Chang, Finding the minimum visible vertex distance between two non-intersecting simple polygons, *Proceedings of the second annual symposium on computational geometry (SoCG)* (1986), 34–42.